X-NET Cross-Network Debugger User's Manual

Packet Radio Temporary Note 141

April 29, 1975

M. Beeler

Digitized by the Internet Archive
in 2025 with funding from
Amateur Radio Digital Communications, Grant 151

https://archive.org/details/temporary-note-141

INTRODUCTION

This document is intended as a user manual for the cross-network debugger, X-NET. X-NET uses a large computer to talk with a small computer across a communications network. The program being debugged is in the small computer, which also runs a compact debugger process which performs examine, deposit, start, stop, etc. commands upon the subject process. The large computer has a large memory and greater processing power, facilitating several debugging conveniences. For example, the user may wish to print the contents of a location as a machine instruction. He requests this by typing a command on his terminal, which is connected to the large computer. The X-NET process in the large computer interprets his command, and transmits an examine command to the small computer's debugger process. This process picks up the desired data and sends it back to the large computer, where it is decoded into instruction mnemonics and printed on the user's terminal.

In our case, the large computer is a PDP-10 running X-NET as a job under the TENEX operating system; the small computer is a PDP-11 running the debugger and user processes under ELF; and the network is the ARPA network. There are applications where the "target machine" might not be a small PDP-11, but some large computer instead. We shall use the more general cross-network debugging term, "target machine" and "PDP-11" interchangeably in this document.

The format of commands typed by the user has similarities to both the PDP-10 DDT and the PDP-11 RUG debuggers. Additional commands control use of the network: connecting to a PDP-11, creating a process therein, printing statistics of network performance, etc. The format of commands and responses between the two computers, which is invisible to the user, was specified by Eric Mader [MADER 1974].

Below is a concise summary of commands at the user's disposal. Following that is a detailed discusssion of the operation of each command. Note that the dollar sign ($) has been used to indicate the "escape" or "alt mode" character except where otherwise noted.

* *

X-NET COMMAND SUMMARY

I Expressions

<number>	octal integer
<number>.	decimal integer
'<character>	ASCII
"<ch1><ch2>	ASCII in low, high bytes of word
&<delim><ch1><ch2><ch3><delim>	radix-50
.	current location

```
$Q                  value of current quantity
$M                  location where mask is stored
<arg>$F             location where <arg>th address offset is stored
+ - * !             arithmetic operators
<byte1>,,<byte0>          two bytes of word for deposit or =
```

II Typeout Mode Setting Commands

```
$A      temporary ASCII
$$A     permanent ASCII
$C      temporary constants
$$C     permanent constants
$I      temporary instruction
$$I     permanent instruction
$R      temporary radix-50
$$R     permanent radix-50
$$D     decimal radix
$$O     octal radix
```

III Target Machine Action Commands

```
<cr>    deposit if argument typed and a location is open
<lf>    as <cr>, then open next location (next instr in instr mode)
^       as <cr>, then open previous location
<arg>/  open location specified by argument
<off#>,<arg>/   open location specified by argument and offset number
      <loc>$B   set breakpoint at location <loc>
<loc>,<num>$B   set breakpoint number <num> at location <loc>
    0,<num>$B   remove breakpoint number <num>
         $$B    remove all breakpoints
<loc>$G start user program at location <loc>
$H      halt user program
$P      proceed last breakpoint trap or resume last halt
<num>$P proceed last breakpoint trap <num> times without typing
$$P     proceed last breakpoint trap indefinitely
            <arg>$W     word search all of memory for <arg>
            <arg>$N     search all of memory for non-word <arg>
<bot>,<top>,<arg>$W     word search locations <bot> to <top>
<bot>,<top>,<arg>$N     non-word search locations <bot> to <top>
                $$Z     set all memory to zero
            <arg>$$Z    set all memory to <arg>
      <bot>,<top>$$Z    set memory from <bot> to <top> inclusive to zero
<bot>,<top>,<arg>$$Z    set memory from <bot> to <top> incl to <arg>
;;Y     load ("yank") disk file into target machine
;;M     merge disk file into target machine
;;U     write ("unget") target machine memory onto disk file
<bot>,<top>;;U  write memory from <bot> to <top> inclusive onto disk
```

IV Miscellaneous Commands

```
rubout  ignore last typein
=       type current quantitiy as number in current radix
:       comment; ignore through next cr
```

```
;V      type X-NET version
;?      type/change run status of program
```

V Buffer Manipulation Commands

```
;I      select another (or <arg>th) setting of examine/deposit switch
;Y      load ("yank") disk file into buffer
;M      merge disk file into buffer
;U      write ("unget") buffer onto disk file
<bot>,<top>;U   write buffer from <bot> to <top> inclusive to disk file
;L      load buffer into target machine core memory
<bot>,<top>;L   load buffer from <bot> to <top> inclusive into core
;R      retrieve target machine core memory into buffer
<bot>,<top>;R   retrieve core from <bot> to <top> inclusive into buffer
```

VI Network Commands

```
?                   print (and optionally clear) statistics
;N                  enter network/process management
<any character>     type status of outstanding network message
<control-A>         abort awaiting network message, abort $N and $W searches
```

* *

COMMAND DETAILS

I Expressions

<number> octal integer

Digits 0 through 9 may be used; 8 and 9 are shorthand for octal 10 and 11.

<number>. decimal integer

'<character> ASCII

The ASCII value of the character following the single quote is the value of this term.

"<ch1><ch2> ASCII in both bytes of word

The value of this term is the ASCII value of the first character in the low byte, plus the ASCII value of the second character in the high byte. It is equivalent to '<ch2>*400+'<ch1> except that "<ch1><ch2> is parsed all as one term.

&<delim><ch1><ch2><ch3><delim> radix-50

<delim> is any character other than <ch1>, <ch2> or <ch3>. The value of this term is the radix-50 value of the string, <ch1><ch2><ch3>. The radix-50 code used for the PDP-11 is:

0	space
1 - 32	A - Z
33	$ (dollar sign)
34	. (period)
35	% (percent sign)
36 - 47	0 - 9

The assembled value for &/ABC/ is 1*50*50 + 2*50 + 3 . Missing argument characters are treated as spaces, so &/A/ = &/A / = &/A / = 1*50*50 .

. current location

Dot (period) has the value of the address most recently opened (see examine commands, below) location.

$Q value of current quantity

In general, $Q (escape Q) has the value of the last expression typed out by X-NET. When that expression was a multi-word instruction, $Q has the value of the first word of that instruction.

$M location of mask

$M (escape M) is the address within the debugger where the mask for word and non-word searches is stored. One-bits stored in $M indicate that those bit positions are to be compared during searches. Initially $M contains -1 (all bits).

<arg>$F location where <arg>th offset is stored

When a program is assembled as several separate pieces and later combined by a linking loader, the loader must relocate addresses in each piece to correspond with their ultimate positions in memory. Often, however, the only listing available contains addresses before this relocation. In this case, it is useful to specify addresses for the debugger to examine as a "relocation quantity" plus the unrelocated address found in the listing. X-NET provides this through eight "relocation registers" which are addressed as 0$F through 7$F. To examine or change the value stored in any of these, type "n$F/". New contents may be typed and then deposited with a carriage return, linefeed or up arrow.

For example, if a program piece has been relocated by 1234, 1234 may be placed in some convenient relocation register n. Then, for instance, typing n,100/ examines location 100 in that piece. (The function of the relocation registers mimics that in DEC's ODT debugger, but the semantics are slightly different. The mnemonic "F", for "offset", was chosen to avoid conflict with X-NET's $R, $$R and ;R commands.)

+ - * ! arithmetic operators

Multiplication (*) and division (!) do not have precedence over addition (+ or space) and subtraction (-). Expressions are evaluated as typed, from left to right. Thus, 1+1*2 = 4.

<byte1>,,<byte0> two bytes of word

Has the value (<byte1>&377)*400+(<byte0>&377), for an argument to deposit and = commands only.

II Typeout Mode Setting Commands

$A temporary ASCII

If a location is open, its contents are retyped as two ASCII bytes. To clarify order, they are typed as ´<charB>,,´<charA>. Either that string or "AB may be used to deposit the same value.

$C temporary constants

If a location is open, retype its contents as an octal or decimal integer.

$I temporary instruction

If a location is open, retype its contents as a PDP-11 instruction. (Currently, 11/40 FPU instructions are included, but 11/45 floating point instructions are not.)

$R temporary radix-50

If a location is open, retype its contents as three characters in radix-50 notation (see above). The same format is used for typing in and typing out radix-50 values.

If no location is open, $A, $C, $I and $R have no effect.

$$A permanent ASCII
$$C permanent constants
$$I permanent instruction
$$R permanent radix-50

Set typeout mode to ASCII, constants, instruction or radix-50, as above. Mode remains in effect until another mode is selected.

$$D decimal radix

Numbers typed out will be base ten, and followed by a decimal point.

$$O octal radix

Numbers typed out will be base eight.

III Target Machine Action Commands

The commands in this section are the principal contributors to computer-computer communication across the communications network. In these commands, some operation is to be performed on the user process. If the operation is complex, X-NET breaks it down into simpler actions compatible with the network debugging protocol [MADER 1974] and the nature of the communications network. An example is breaking a "clear memory" command into several small blocks.

<cr> deposit

Deposit argument in open location if an argument precedes the carriage return and a location is open. Otherwise, no effect.

<lf> deposit, examine next

As <cr>, then open the next location. In instruction mode, the location after all of the (possibly multi-word) instruction is opened.

^ deposit, examine previous

As <cr>, then open previous location.
Circumflex appears as up arrow on some terminals.

<arg>/ examine

Open the location specified by the argument, <arg>.

Carriage return, line feed, up arrow and slash operate as in such debuggers as PDP-10 DDT and PDP-11 RUG. (Currently, there is no byte-by-byte mode in X-NET; only full words may be examined. X-NET also currently lacks commands to open a location addressed by the last location opened.)

<off#>,<arg>/ examine

Like <arg>/ but first adds the contents of the <off#>th offset register to <arg>. Equivalent to "<off#>$F/ $Q+<arg>/". See discussion above under $F command.

<loc>$B set breakpoint at location <loc>

<loc> must be even. A breakpoint may not be set at location 0, nor where one is already set. A message is printed when a breakpoint is hit. (Currently, there is a limit of 8 breakpoints set simultaneously. X-NET does not have "conditional" breakpoints, nor "actions" associated with breakpoints to be performed when a break occurs.)

<loc>,<num>$B set breakpoint number <num> at location <loc>

As above, but a particular one of the breakpoints available is specified. (Currently, <num> must be from 0 to 7, inclusive.) (Note a difference in command syntax from PDP-10 DDT: <loc>$<num>B will NOT work.)

0,<num>$B remove breakpoint number <num>

$$B remove all breakpoints

<loc>$G start user program at location <loc>

The network debugging protocol specifies different commands to make the program run when it is stopped at a breakpoint ("proceed" it) or when it is otherwise stopped ("resume" it). $G sends the appropriate command in either case. Execution will commence at location <loc> in either case.

$P proceed (or resume, as appropriate) last halt or breakpoint trap

Execution commences from where it last stopped.

<num>$P proceed last breakpoint trap <num> times

This command may be used to reinitiate execution only after a breakpoint trap. It instructs the debugger process in the PDP-11 to ignore <num> breaks of that breakpoint. On the <num>+1th break, the PDP-11 informs the PDP-10 of the break, and only then is a breakpoint message typed on the user's terminal. Implementing "multiple proceeds" in the PDP-11 this way makes them faster.

$$P proceed last breakpoint indefinitely

The last stop must have been a breakpoint trap. The breakpoint is proceeded. With each subsequent occurrence of this particular breakpoint trapping, a message is typed on the user's terminal and a command is automatically sent to the PDP-11 to proceed. Quitting from this mode is accomplished by halting the user process ($H) and removing the breakpoint (0,<num>$B).

$H halt user program

<arg>$W word search

All of memory is searched for words which, when and-ed with the mask stored in $M, equal <arg>. The location and contents of each such word are typed. Control-A aborts the search. "All of memory" means from 0 to a value X-NET believes to be the highest existing core address (see ;V command).

<arg>$N non-word search

Same as above, but location and contents are printed only if word, when and-ed with the mask, does NOT equal <arg>.

<bot>,<top>,<arg>$W word search between limits
<bot>,<top>,<arg>$N non-word search between limits

Same as word ($W) and non-word ($N) searches above, but only memory locations <bot> through <top>, inclusive, are searched.

$$Z set all memory to zero

Places zero in all memory locations. "All" means from 0 to X-NET's idea of the last core location (see ;V command).

<arg>$$Z set all memory to <arg>

Same as $$Z, but <arg>, instead of zero, is placed in each location.

<bot>,<top>$$Z zero memory between limits

Same as $$Z, but only locations <bot> through <top>, inclusive, are set to zero. <bot> and <top> must be even.

<bot>,<top>,<arg>$$Z set memory between limits to <arg>

Memory locations <bot> through <top>, inclusive, are loaded with <arg>. This is done on a word-by-word basis, not byte-by-byte. <bot> and <top> must be even.

;;Y load ("yank") disk file into target machine

A TENEX file name is requested, and the format in which the file was written. All memory in the user process is loaded with zero. The transfer is then made. R7 is set to the starting address given in the file, if any.

;;M merge disk file into target machine

Same as ;;Y, but memory is not zeroed before the transfer.

;;U write ("unget") target machine memory onto disk file

A TENEX file name is requested, and the format in which the file is to be written. Pairs of addresses are requested. As each pair is given, locations between them (inclusive) are written as a block in the disk file. Finally, a start address is requested and written.

<bot>,<top>;;U write target machine memory between limits onto disk

Same as ;;U, but only the block of locations from <bot> to <top>, inclusive, is written. Further pairs of addresses are not requested.

IV Miscellaneous Commands

rubout ignore last typein

Rubout (delete) causes a partially typed command to be ignored.

= type as number

Causes the value of its argument (or $Q, if no argument is given) to be printed as an octal or decimal (see radix setting commands above) integer.

: comment

The colon and following characters through and including the next carriage return will be ignored. This is useful for conversing with another user whose terminal is copying the characters printed on your terminal.

;V type X-NET version number

X-NET types its vintage and the value of two of its internal parameters: the maximum allowed number of data bytes per network message (MXBYTS), and the highest address corresponding to PDP-11 core memory (MEMTOP). Also permits resetting MEMTOP.

;? type/change run status of program

In certain cases X-NET may be mistaken about whether the user program is running. One such case is when X-NET starts debugging a process it did not create. In this case, X-NET arbitrarily assumes the process is halted. Should the user happen to know otherwise, the ;? command may be used to alter X-NET's idea of the program's state. The ;? command is also useful when the user has lost track of his program's status. Also permits forgetting about breakpoints that are set.

V Buffer Manipulation Commands

The user can examine, deposit, patch, etc. either on the target machine or on a buffer which he can manipulate. The user can select either the target machine or the buffer for such actions. This mode setting command is ;I and is described below. When attending to the target machine, the commands ;;Y and ;;M are used to transfer data from a file to the target machine, and the command ;;U is used to transfer data in the reverse direction. When attending to the buffer, the commands ;Y and ;M are used to transfer data from a file into the buffer, the command ;U to transfer data from buffer to file, the command ;L to load the target machine from the buffer, and ;R to load the buffer from the target machine.

IMPLEMENTATION:

X-NET maintains in the PDP-10 a copy (cache) of data in the PDP-11. The greatest advantage this has is to speed up response to most examine and deposit commands. Consider a case where the user has halted his PDP-11 program and now wishes to fish around for signs of a bug. He examines a particular memory location in his program. X-NET, however, requests not only the contents of that particular location, but of a larger block of memory which includes that location. Responses to the user's examine commands for other locations within that block can now be made rapidly, since no network or PDP-11 interaction is needed. Similarly, the user may deposit new values into locations within this block. Each deposit command need not be transmitted to the PDP-11 immediately. Rather, each such location may be updated in the cache local to the PDP-10, and marked in some way as changed. The user may start his program running again, at which time X-NET will first transmit all of these deferred changes. After propagating any changes to the PDP-11, X-NET starts the user program as requested. Deferring changes in this manner allows not only faster response to individual deposit commands, but also more efficient propagation of such changes to the PDP-11, since entire blocks of changed locations can be sent in single network messages.

The user may desire two other modes of cache operation besides the deferred mode described above. The deferred mode is the most simple for the user, since the existence of the cache is apparent to him only as an improvement in response time to examine and deposit commands (and as a small increase in the delay of commands which start his program running). The existence of the cache is always invisible to the user except for commands to enable and disable its use. (It is initially enabled.) On the other hand, values stored in the local cache are out of date if the process is running; or if a second PDP-11 process is writing in the same address space; or if direct memory access (DMA) devices are writing into the address space. In the first case, X-NET is aware that the process is running, and takes appropriate action. In the latter two cases, the user needs an "immediate" mode of the examine and deposit commands, to bypass any stale values sequestered in the cache. In the immediate mode, each examine command fetches a fresh value from the PDP-11, and each deposit command places its value in the PDP-11 before the next command is accepted.

The third mode of examine and deposit commands operates entirely local to the PDP-10. The PDP-11 is not referenced for examine or deposit commands. (Changed locations are so marked, to allow later propagation of these changes to the PDP-11 if the user later selects immediate or deferred mode, or if he later starts his program.) This "local" mode is principally used to make patches and format conversions to disk files. As an example, an "LDA" format file consisting of many small load blocks may be read into the local cache; a few patches may be made; and the result may be written out as a disk file in "BIN" format of one large block (and hence very efficient for later loading).

;I select another setting of "I/D/L E/D switch"
<arg>;I select <arg>th setting of "I/D/L E/D switch"

The "immediate/deferred/local examine/deposit switch" selects among three modes of X-NET internal buffer operation, as described above. <arg>, if supplied, must be 0, 1 or 2. If immediate mode (<arg> = 0) is selected, all buffer locations marked as changed are propagated to the PDP-11 before the next X-NET command is accepted. ;I with no argument sequences through immediate, deferred (the initial mode) and local modes.

;Y load ("yank") disk file into buffer

A TENEX file name is requested, and the format in which the file was written. The buffer is filled with zeros. The transfer is made. R7 is set to the starting address given in the file, if any. The lowest and highest addresses transferred are remembered; a subsequent ;L command (see below) with no arguments will default to these addresses.

;M merge disk file into buffer

Same as ;Y, but buffer is not zeroed before the transfer. Note that commands to write a disk file (;U and ;;U) and to load the PDP-11 (;L and ;;L) always transfer entire PDP-11 words, while commands to read a disk file (;Y ;M ;;Y and ;;M) accept blocks of odd numbers of bytes as well. For example, a file ;Y reports as extending from (bytes) 100 to 105 is written completely by a ;U command with bounds 100 to 104.

;U write ("unget") buffer onto disk file

A TENEX file name is requested, and the format in which the file is to be written. Pairs of addresses are requested. (Use comma to separate the arguments in each pair.) As each pair is given, locations between them (inclusive) are written as a block in the disk file. Finally, a start address is requested and written.

<bot>,<top>;U write buffer between limits onto disk file

Same as ;U, but only the block of locations from <bot> to <top>, inclusive, is written. Further pairs of addresses are not requested.

;L load buffer into target machine core

Same as <bot>,<top>;L (see below), but the limits <bot> and <top> are defaulted to the lowest and highest addresses in the most recent ;Y or ;M command (see above).

<bot>,<top>;L load buffer between limits into target machine core

The contents of the buffer between locations <bot> and <top>, inclusive, are deposited in the PDP-11. The user is then asked whether to also deposit the local copy of the state vector, which is the eight general registers and the processor status word.

;R retrieve target machine core into buffer

Makes a copy in the buffer of all PDP-11 core locations, from 0 to X-NET's idea of the last core location (see ;V command).

<bot>,<top>;R retrieve target machine core between limits into buffer

PDP-11 core locations <bot> through <top>, inclusive, are copied into the buffer.

VI Network Commands

? print (and optionally clear) statistics

X-NET keeps statistics on the nature of its use of the ARPA network, and on the response time the ARPA network has exhibited. Currently, "message" in this printout means a message sent by X-NET, while "size" means total network traffic including both the bytes in X-NET's commands and those in the replies.

;N enter network/process management

When computation is distributed among several computers, it becomes important to have simple means to unambiguously determine what inter-computer connections are in progress, and means to change these easily. X-NET is relatively uncomplicated in this regard; the user's terminal is always logically connected to the PDP-10 X-NET process, which is always either connected to no PDP-11 or to one specific PDP-11 (ARPA network host). If connected to a host, X-NET is either debugging no process in it or is debugging one specific process in it. Despite the simplicity of this distributed computation, the author feels that over-explicit management of network and process connections is desirable.

The ;N command enters a mode in which certain commands become available, instead of and to the exclusion of, other X-NET commands. The network and process management commands are:

Call host	End debug
Create process	Quit management
Debug process	Release net
Destroy process	Verify life of host

They are invoked by typing their first letter. Most are self-explanatory; "Release net" optionally does an "End debug" if a process is being debugged, and then releases various TENEX resources associated with this implementation of cross-network debugging. "Verify life of host" sends a no-operation message to the debugger process in the PDP-11 and awaits the response.

<any character> type network status

Each message sent by X-NET to the PDP-11 should elicit a response from the debugger process in the PDP-11. While X-NET is awaiting such a response, typing any character causes a printout of what operation was requested by the outstanding message, and how much time has elapsed since the message was given to TENEX to transmit.

<control-A> abort

Aborts out of awaiting a network message. Any command which occasioned this awaiting is aborted. X-NET returns to its top level command scanner. Control-A is also used to abort out of word and non-word searches.

VII Possible Extensions to X-NET

INSTRUCTION TYPEIN
This is probably the first improvement that should be made to X-NET.

SYMBOLS
Any debugger not under severe core restraints has no excuse to not be symbolic. Writing a symbolic debugger for the PDP-11 is complicated by multiplicity of symbol table formats. Some PDP-11 assemblers produce none; some produce a symbol table only as part of a listing file; and so forth.

SINGLE STEP
Although the PDP-11 is capable of single stepping (executing one instruction and then returning to the debugger), and the current debugging protocol includes such a feature, neither the PDP-10 nor the PDP-11 end of X-NET has this feature.

AUTOMATIC TYPEOUT MODE
An ingenious mechanism devised by Eric Mader of BBN could be employed to cause memory contents to almost always be typed out in the mode desired by the user. The assembler would emit extra "mode bits" with each word (or byte) of compiled code, specifying the mode of the source text which generated that code. The debugger would then reference these to set the (default) typeout mode.

EXAMINE PREVIOUS INSTRUCTION
As a corollary to the automatic typeout mode, the assembler could also emit a bit marking the first word of instructions. This would be the most direct of several schemes proposed to accomplish the "examine previous" function in instruction mode.

BDDT
Plans at BBN include fusing parts of the debugger for the BCPL language, called BDDT, with X-NET, resulting in a cross-network BDDT. This would permit easier debugging of BCPL code running in a PDP-11 on the ARPA network.

OPEN SOURCE, DESTINATION, ADDRESSED LOCATION
Another debugger (RUG) has the capability of opening the source or destination of the instruction currently open, as well as the word (or byte) whose address is the current value. These useful features could be incorporated in X-NET. Examining and depositing by bytes would be a natural concomitant to this.

EXAMINE STATE VECTOR
The current debugging protocol does not provide this function; hence registers and processor status word of the user's process cannot be examined while the process is running. Implementation of features such as this requires very careful interaction with whatever operating system is resident in the PDP-11.

QUERY STATE
Here "state" means whether the process is running or halted, and perhaps other parameters specific to the operating system in use. Again, the current debugging protocol does not provide this function. Although there exists a danger of timing errors (e.g., the process stopping after the debugger tells the user it's running), there exist situations where a user begins debugging a pre-existing process and this feature would provide the most reliable state data.

SEPARATE CACHE AND BUFFER
The X-NET concept of a buffer of simulated PDP-11 memory is implemented as the same internal area used for the cache. These two could be completely separate. The issues involved are complex and beyond the scope of this discussion, although the decision to implement only one internal area in X-NET was made after consideration of them.

FPP REGISTERS
Another debugger (RUG) has a mechanism for examining the accumulators of the PDP-11/45 Floating Point Processor option. These do not appear as Unibus addresses, so such a mechanism is quite useful to FPP programmers. X-NET could also be extended to know FPP instruction mnemonics.

SET MEMORY ENTENSION
Neither the current protocol nor the X-NET software provide means to examine physical core addresses higher than 16 bits (32 K bytes). A command to specify the value of the 17th and 18th address bits would accomplish this.

* *

CAVEAT

A known bug exists in handling responses from the PDP-11 when one (or more) breakpoints are being automatically proceeded, and the user types a target machine action command. When an asynchronous reply, "breakpoint halt," arrives from the PDP-11, X-NET sends a "proceed" command without user intervention, and awaits the "proceeding" reply from the PDP-11. If, however, the user has typed an examine command to X-NET, then X-NET will be awaiting the response to the examine. When the asynchronous breakpoint halt message arrives, X-NET recurses a level and sends the proceed command. Unfortunately, X-NET's messaage receiving code is not structured to be simultaneously awaiting both the proceed response and the examine response. Consequently, the examine response will arrive and be discarded as an "ASYNCHRONOUS REPLY, MYSTERIOUS OPCODE 6." The proceed reply will then arrive, and X-NET will return to awaiting the response to the examine command. Currently, the user's only recourse at this point is to abort out of the hung command and try it again. Halting a process which is in a tight loop with an auto-proceed breakpoint in the loop could thus prove difficult. Until this difficulty has been fixed, beware.

* *

REFERENCES

MADER 1974
"Network Debugging Protocol," Eric Mader, Request For Comments #643, Bolt Beranek and Newman division 52, 50 Moulton St., Cambridge, Mass. 02138, July 1974

DDT "DEC System 10 Assembly Language Handbook," Program Library, DEC, Maynard, Mass. 01754

DDT and TENEX
"TENEX User's Guide," Bolt Beranek and Newman, 50 Moulton St., Cambridge, Mass. 02138

ELF "ELF System Programmer's Guide," D. L. Retz, Speech Communications Research Lab, 800A Miramonte Dr., Santa Barbara, Cal. 93109

RUG RUG is a symbolic, machine language debugger for the PDP-11, written and in use at MIT's Artificial Intelligence Lab, 545 Tech Square, Cambridge, Mass. 01238. It is usally assembled to use a disk to swap user programs and itself. Only a small resident portion remains in core while the user program runs. It has instruction typein and typeout, byte-by-byte examining, symbols obtained from PDP-10 files written by the Lab's cross-machine assembler, single instruction stepping, floating point register examining, awareness of memory management and processor modes, etc.